Claude Code Workflow

사용 원칙 및 워크플로 가이드

Claude Code를 효과적으로 사용하기 위한

핵심 원칙과 생산성 높은 워크플로 구축 방법

https://github.com/krdn/claude-code-workflow

목차

홈

Claude Code Workflow

Claude Code의 핵심 사용 원칙과 생산성 높은 워크플로 구축 방법에 대한 연구 저장소입니다.

개요

이 Wiki에서는 Claude Code를 효과적으로 사용하기 위한 다양한 가이드와 예제를 제공합니다.

학습 로드맵

난이도별 학습 경로입니다. 순서대로 따라가면 Claude Code를 체계적으로 익힐 수 있습니다.

입문

순서	문서	내용
1	시작하기	설치, 기본 사용법, CLAUDE.md 설정

기본

순서	문서	내용
2	핵심 원칙	7가지 사용 원칙 (컨텍스트, 분할, 검증 등)
3	워크플로 선택	코드 리뷰 · 디버깅 · 리팩토링

고급

순서	문서	내용
4	모범 사례	MCP, Hooks, 팀 협업
5	빠른 참조	자주 쓰는 프롬프트, 설정 요약

```
입문 (15분) → 기본 (30분) → 고급 (필요시)
```

빠른 시작

- 시작하기 - Claude Code 설치 및 기본 사용법
- 핵심 원칙 - 효과적인 사용을 위한 핵심 원칙
- 모범 사례 - 생산성을 높이는 팁

주요 섹션

문서

문서	설명
시작하기	Claude Code 설치 및 기본 사용법
핵심 원칙	효과적인 사용을 위한 핵심 원칙
모범 사례	생산성을 높이는 팁

워크플로

워크플로	설명
코드 리뷰	PR 리뷰 및 피드백
디버깅	버그 분석 및 수정
리팩토링	코드 개선 및 구조화

예제

예제	설명
Hooks 예제	자동화 훅 설정
MCP 서버 예제	MCP 서버 설정
프롬프트 예제	효과적인 프롬프트

프롬프트 패턴

패턴	설명
코드 분석	코드 분석 프롬프트
코드 작성	코드 작성 프롬프트
버그 수정	버그 수정 프롬프트
리팩토링	리팩토링 프롬프트

패턴	설명
테스트	테스트 작성 프롬프트

템플릿

템플릿	설명
CLAUDE.md 템플릿	프로젝트별 CLAUDE.md

참고 자료

- Claude Code 공식 문서
- GitHub 저장소

시작하기

시작하기

> 학습 경로: 시작하기 → 핵심 원칙 → 모범 사례 → 빠른 참조 | ↑ Home

Claude Code를 효과적으로 사용하기 위한 기본 가이드입니다.

설치

```
npm install -g @anthropic-ai/claude-code
```

기본 사용법

```
# 현재 디렉토리에서 Claude Code 시작
claude

# 특정 프롬프트와 함께 시작
claude "이 프로젝트의 구조를 분석해줘"

# 특정 파일에 대해 질문
claude "src/main.ts 파일을 분석해줘"
```

CLAUDE.md 설정

프로젝트 루트에 CLAUDE.md 파일을 생성하여 프로젝트 컨텍스트를 제공하세요.

```
# 프로젝트 개요
이 프로젝트는 ...

## 기술 스택
- Node.js 20
- TypeScript 5.x
- React 18

## 빌드 및 테스트
- 빌드: `npm run build`
- 테스트: `npm test`

## 코딩 컨벤션
- ESLint + Prettier 사용
- 함수형 컴포넌트 선호
```

주요 기능

1. 코드 분석

```
claude "이 함수의 로직을 설명해줘"
claude "이 코드에서 성능 문제가 있을 수 있는 부분을 찾아줘"
```

2. 코드 작성

```
claude "사용자 인증 기능을 구현해줘"
claude "이 API에 대한 테스트 코드를 작성해줘"
```

3. 버그 수정

```
claude "이 에러를 수정해줘: [에러 메시지]"
claude "빌드 에러를 분석하고 수정해줘"
```

4. 리팩토링

```
claude "이 코드를 더 읽기 쉽게 리팩토링해줘"
claude "중복 코드를 제거해줘"
```

효과적인 프롬프트 작성

좋은 예시

```
"src/components/Button.tsx 파일에서 onClick 핸들러에
디바운싱을 적용해줘. 딜레이는 300ms로 설정해."
```

개선이 필요한 예시

```
"버튼 고쳐줘"
```

다음 단계

- 핵심 원칙 - Claude Code 사용 핵심 원칙
- 모범 사례 - 생산성을 높이는 모범 사례

핵심 원칙

핵심 사용 원칙

> 학습 경로: 시작하기 → 핵심 원칙 → 모범 사례 → 빠른 참조 | ↑ Home

Claude Code를 효과적으로 사용하기 위한 핵심 원칙들입니다.

1. 명확한 컨텍스트 제공

CLAUDE.md 활용

프로젝트 루트에 CLAUDE.md 파일을 생성하여 프로젝트 컨텍스트를 제공합니다.

```
# 프로젝트명

## 개요
프로젝트 설명...

## 기술 스택
- 언어/프레임워크
- 데이터베이스
- 인프라

## 빌드/테스트 명령어
npm run build
npm test

## 코딩 컨벤션
- 네이밍 규칙
- 파일 구조
```

계층적 CLAUDE.md

- 전역: ~/.claude/CLAUDE.md
- 프로젝트: ./CLAUDE.md
- 디렉토리: ./src/CLAUDE.md

2. 점진적 작업 분할

큰 작업을 작은 단위로

```
# 좋지 않음
"전체 인증 시스템을 구현해줘"

# 좋음
"1. 먼저 로그인 API 엔드포인트를 구현해줘"
"2. JWT 토큰 생성 로직을 추가해줘"
```

```
"3. 미들웨어에서 토큰 검증을 구현해줘"
```

TodoWrite 활용

Claude Code의 TodoWrite 도구를 사용하여 작업을 추적합니다.

3. 검증 주기 유지

변경 후 확인

```
# 코드 변경 후
"빌드해서 에러가 없는지 확인해줘"
"테스트를 실행해줘"
"타입 체크를 실행해줘"
```

점진적 커밋

작은 단위로 커밋하여 롤백이 용이하도록 합니다.

4. 효과적인 프롬프트 작성

구체적으로 작성

```
# 좋지 않음
"코드 수정해줘"

# 좋음
"src/utils/date.ts의 formatDate 함수에서
타임존을 KST로 변환하는 로직을 추가해줘"
```

맥락 포함

```
# 좋지 않음
"에러 수정해줘"

# 좋음
"npm run build 실행 시 다음 에러가 발생해:
[에러 메시지]
이 에러를 수정해줘"
```

제약 조건 명시

```
"React 18의 새로운 훅을 사용하지 않고 구현해줘"
"외부 라이브러리 추가 없이 구현해줘"
```

5. 코드 이해 후 수정

읽기 우선

Claude Code가 코드를 수정하기 전에 반드시 해당 코드를 읽도록 합니다.

```
"먼저 이 파일을 분석하고, 그 다음 수정해줘"
```

영향 범위 파악

```
"이 함수를 수정하면 어떤 파일들이 영향을 받는지 분석해줘"
```

6. 보안 의식

민감 정보 보호

- API 키, 비밀번호 하드코딩 금지
- .env 파일 커밋 금지
- 환경 변수 사용

코드 검토

```
"이 코드에 보안 취약점이 있는지 검토해줘"
```

7. 문서화

코드 변경 시 문서 업데이트

```
"이 변경사항을 README에 반영해줘"
```

ADR (Architecture Decision Record)

중요한 기술 결정은 문서화합니다.

요약

원칙	핵심
컨텍스트	CLAUDE.md로 프로젝트 정보 제공
분할	큰 작업을 작은 단위로
검증	변경 후 빌드/테스트 확인
프롬프트	구체적이고 명확하게
이해	수정 전 코드 분석
보안	민감 정보 보호

원칙	핵심
문서화	변경사항 기록

모범 사례

모범 사례

Claude Code를 사용한 생산성 높은 개발을 위한 모범 사례입니다.

세션 관리

컨텍스트 유지

- 관련 작업은 같은 세션에서 수행
- 새 세션 시작 시 필요한 컨텍스트 제공

세션 정리

```
# 대화 내용 요약 저장
claude "/context save"
```

MCP 서버 활용

데이터베이스 연동

```
{
  "mcpServers": {
    "postgres": {
      "command": "mcp-server-postgres",
      "args": ["postgresql://..."]
    }
  }
}
```

외부 API 연동

브라우저, 파일 시스템 등 다양한 MCP 서버 활용

자세한 설정 예제는 MCP 서버 예제를 참조하세요.

훅 자동화

Pre-commit 훅

커밋 전 자동으로 린트, 테스트 실행

빌드 훅

코드 변경 후 자동 빌드 확인

자세한 훅 예제는 Hooks 예제를 참조하세요.

팀 협업

공유 CLAUDE.md

팀 전체가 사용하는 공통 컨텍스트 파일 관리

코드 리뷰 워크플로

PR 리뷰에 Claude Code 활용

자세한 내용은 코드 리뷰 워크플로를 참조하세요.

학습 및 개선

실패에서 배우기

작동하지 않은 프롬프트 분석 및 개선

패턴 축적

효과적인 프롬프트 패턴을 문서화

다양한 프롬프트 패턴은 프롬프트 예제에서 확인할 수 있습니다.

빠른 참조

빠른 참조 (Cheatsheet)

Claude Code 사용 시 빠르게 참조할 수 있는 요약 문서입니다.

자주 사용하는 프롬프트

코드 분석

상황	프롬프트 예시
구조 분석	이 파일의 구조를 분석해줘
함수 이해	이 함수의 로직을 설명해줘
의존성 파악	이 모듈이 의존하는 파일들을 찾아줘
성능 분석	이 코드에서 성능 문제가 있을 수 있는 부분을 찾아줘

→ 더 많은 코드 분석 프롬프트

버그 수정

상황	프롬프트 예시
에러 분석	이 에러를 분석하고 수정해줘: [에러 메시지]
빌드 에러	빌드 시 다음 에러가 발생해: [에러]. 수정해줘
테스트 실패	이 테스트가 실패하는 원인을 분석하고 수정해줘
런타임 에러	npm run dev 실행 시 다음 에러가 발생해: [에러]

→ 더 많은 버그 수정 프롬프트

리팩토링

상황	프롬프트 예시
함수 분리	이 긴 함수를 작은 함수들로 분리해줘

상황	프롬프트 예시
중복 제거	중복된 코드를 공통 함수로 추출해줘
타입 개선	any 타입을 구체적인 타입으로 변경해줘
네이밍	변수명과 함수명을 더 명확하게 변경해줘

→ 더 많은 리팩토링 프롬프트

코드 작성

상황	프롬프트 예시
새 기능	사용자 인증 기능을 구현해줘
테스트	이 API에 대한 테스트 코드를 작성해줘
API 추가	GET /users 엔드포인트를 추가해줘
컴포넌트	버튼 컴포넌트를 React로 만들어줘

→ 더 많은 코드 작성 프롬프트

CLAUDE.md 필수 항목

프로젝트 루트에 CLAUDE.md 파일을 생성하세요.

```
# 프로젝트명

## 개요
프로젝트 설명 (1-2문장)

## 기술 스택
- 언어: TypeScript 5.x
- 프레임워크: React 18
- 데이터베이스: PostgreSQL
- 인프라: Docker, AWS

## 빌드/테스트 명령어
- 설치: `npm install`
- 개발: `npm run dev`
- 빌드: `npm run build`
- 테스트: `npm test`
- 린트: `npm run lint`

## 코딩 컨벤션
- ESLint + Prettier 사용
- 함수형 컴포넌트 선호
- 파일명: kebab-case
- 컴포넌트: PascalCase
- 함수/변수: camelCase

## 주의사항
- API 키 하드코딩 금지
- .env 파일 커밋 금지
- 프로덕션 DB 직접 접근 금지
```

→ CLAUDE.md 템플릿 상세

체크리스트

- [ ] 프로젝트 개요 작성
- [ ] 기술 스택 명시
- [ ] 빌드/테스트 명령어 포함
- [ ] 코딩 컨벤션 정의
- [ ] 보안 주의사항 명시

Hooks 이벤트 요약

~/.claude/settings.json에 설정합니다.

이벤트	용도	예시
PreToolUse	도구 실행 전 검증	커밋 전 린트/테스트 실행
PostToolUse	도구 실행 후 자동화	파일 수정 후 자동 포맷팅
Notification	알림 전송	Slack 알림
Stop	세션 종료 시	리소스 정리

예제: 자동 린트

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "command": "npx eslint --fix ${file}",
        "timeout": 30000
      }
    ]
  }
}
```

예제: 커밋 가드

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "pattern": "git commit",
        "command": "npm run lint && npm test",
        "on_error": "block",
        "timeout": 120000
      }
    ]
```

```
  }
}
```

→ 더 많은 Hooks 예제

MCP 서버 빠른 설정

~/.claude/settings.json에 설정합니다.

PostgreSQL

```
{
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": [
        "@modelcontextprotocol/server-postgres",
        "postgresql://user:pass@localhost:5432/dbname"
      ]
    }
  }
}
```

SQLite

```
{
  "mcpServers": {
    "sqlite": {
      "command": "npx",
      "args": [
        "@modelcontextprotocol/server-sqlite",
        "/path/to/database.db"
      ]
    }
  }
}
```

GitHub

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    }
  }
}
```

→ 더 많은 MCP 예제

유용한 명령어

Claude Code CLI

```
# 기본 시작
claude

# 프롬프트와 함께 시작
claude "이 프로젝트를 분석해줘"

# 특정 파일 분석
claude "src/main.ts 파일을 분석해줘"

# 컨텍스트 저장
claude "/context save"

# 도움말
claude --help
```

자주 쓰는 슬래시 명령어

명령어	설명
/help	도움말 표시
/clear	대화 내용 초기화
/context save	컨텍스트 저장
/compact	대화 압축

관련 문서

- 시작하기 - 기본 설치 및 사용법
- 핵심 원칙 - 7가지 사용 원칙
- 모범 사례 - MCP, Hooks, 팀 협업
- 코드 리뷰 워크플로 - PR 리뷰 프로세스
- 디버깅 워크플로 - 버그 분석 프로세스
- 리팩토링 워크플로 - 코드 개선 프로세스

워크플로: 코드 리뷰

코드 리뷰 워크플로

Claude Code를 활용한 효과적인 코드 리뷰 워크플로입니다.

기본 워크플로

1. PR 분석

```
claude "gh pr view 123의 변경사항을 분석해줘"
```

2. 코드 검토

```
claude "이 PR에서 잠재적인 버그나 문제점을 찾아줘"
```

3. 개선 제안

```
claude "이 코드의 개선점을 제안해줘"
```

4. 리뷰 작성

```
claude "리뷰 코멘트를 작성해줘"
```

체크리스트

- [ ] 코드 스타일 일관성
- [ ] 에러 처리
- [ ] 테스트 커버리지
- [ ] 보안 취약점
- [ ] 성능 영향
- [ ] 문서화

프롬프트 예시

전체 리뷰

```
이 PR의 변경사항을 다음 관점에서 리뷰해줘:
1. 코드 품질
2. 버그 가능성
```

```
3. 성능 영향
4. 테스트 충분성
```

특정 파일 리뷰

```
src/services/auth.ts의 변경사항을 보안 관점에서 검토해줘
```

관련 문서

- 코드 분석 프롬프트 - 더 많은 분석 프롬프트
- 디버깅 워크플로 - 버그 발견 시 후속 작업
- 리팩토링 워크플로 - 코드 개선 시 후속 작업

워크플로: 디버깅

디버깅 워크플로

Claude Code를 활용한 효과적인 디버깅 워크플로입니다.

기본 워크플로

1. 문제 분석

```
claude "다음 에러 메시지를 분석해줘: [에러 메시지]"
```

2. 원인 추적

```
claude "이 에러가 발생하는 코드 경로를 추적해줘"
```

3. 해결책 제안

```
claude "이 문제의 해결 방법을 제안해줘"
```

4. 수정 및 검증

```
claude "수정하고 테스트를 실행해서 확인해줘"
```

디버깅 전략

로그 분석

```
이 로그 파일에서 에러 패턴을 찾아줘
```

스택 트레이스 분석

```
이 스택 트레이스를 분석하고 근본 원인을 찾아줘
```

재현 단계 확인

```
이 버그를 재현하는 단계를 정리해줘
```

프롬프트 예시

런타임 에러

```
npm run dev 실행 시 다음 에러가 발생해:
[에러 메시지]
원인을 분석하고 수정해줘
```

빌드 에러

```
빌드 시 타입 에러가 발생해:
[에러 메시지]
이 에러를 수정해줘
```

테스트 실패

```
이 테스트가 실패하는 원인을 분석하고 수정해줘
```

관련 문서

- 코드 리뷰 워크플로
- 리팩토링 워크플로
- 버그 수정 프롬프트

워크플로: 리팩토링

리팩토링 워크플로

Claude Code를 활용한 효과적인 리팩토링 워크플로입니다.

기본 워크플로

1. 현재 코드 분석

```
claude "이 파일의 코드 구조를 분석해줘"
```

2. 문제점 식별

```
claude "이 코드에서 개선이 필요한 부분을 찾아줘"
```

3. 리팩토링 계획

```
claude "리팩토링 계획을 세워줘"
```

4. 단계적 수정

```
claude "먼저 [특정 부분]부터 리팩토링해줘"
```

5. 테스트 확인

```
claude "리팩토링 후 테스트를 실행해줘"
```

리팩토링 패턴

함수 추출

```
이 긴 함수를 작은 함수들로 분리해줘
```

중복 제거

```
중복된 코드를 공통 함수로 추출해줘
```

네이밍 개선

```
변수명과 함수명을 더 명확하게 변경해줘
```

타입 안전성

```
any 타입을 구체적인 타입으로 변경해줘
```

주의사항

- 리팩토링 전 테스트 확인
- 작은 단위로 점진적 수정
- 각 단계마다 빌드/테스트 확인
- 기능 변경 없이 구조만 개선

프롬프트 예시

전체 리팩토링

```
이 모듈을 클린 코드 원칙에 따라 리팩토링해줘.
단, 기능은 변경하지 않고 구조만 개선해줘.
```

특정 패턴 적용

```
이 코드에 Repository 패턴을 적용해줘
```

관련 문서

- 코드 리뷰 워크플로
- 디버깅 워크플로
- 리팩토링 프롬프트

예제: Hooks

Claude Code Hooks 예제

Claude Code 훅을 활용한 자동화 예제입니다.

훅 개요

Claude Code는 다양한 이벤트에 반응하는 훅을 지원합니다:

이벤트	설명
PreToolUse	도구 실행 전
PostToolUse	도구 실행 후
Notification	알림 발생 시
Stop	작업 완료 시

설정 위치

```
~/.claude/settings.json
```

---

1. 자동 린트 (auto-lint)

파일 수정 후 자동으로 ESLint를 실행합니다.

```
{
  "$schema": "https://claude.ai/schemas/hooks.json",
  "description": "파일 수정 후 자동으로 린트를 실행하는 훅",
  "hooks": {
    "PostToolUse": [
      {
        "matcher": {
          "tool_name": "Edit|Write"
        },
        "hooks": [
          {
            "type": "command",
            "command": "npm run lint:fix -- $CLAUDE_FILE_PATH",
            "description": "수정된 파일에 대해 ESLint 자동 수정 실행",
            "timeout": 30000,
            "on_error": "warn"
          }
        ]
      }
```

```
    ]
  }
}
```

---

2. 커밋 가드 (commit-guard)

git commit 전에 테스트와 린트를 검증합니다.

```
{
  "$schema": "https://claude.ai/schemas/hooks.json",
  "description": "git commit 전에 테스트와 린트를 검증하는 훅",
  "hooks": {
    "PreToolUse": [
      {
        "matcher": {
          "tool_name": "Bash",
          "command_pattern": "git commit"
        },
        "hooks": [
          {
            "type": "command",
            "command": "npm run lint && npm test",
            "description": "커밋 전 린트 및 테스트 실행",
            "timeout": 120000,
            "on_error": "block",
            "error_message": "린트 또는 테스트 실패. 커밋이 차단되었습니다."
          }
        ]
      }
    ]
  }
}
```

---

3. Slack 알림 (notification-slack)

작업 완료 및 에러 발생 시 Slack으로 알림을 보냅니다.

```
{
  "$schema": "https://claude.ai/schemas/hooks.json",
  "description": "작업 완료 시 Slack으로 알림을 보내는 훅",
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "curl -X POST -H 'Content-type: application/json' --data '{\"text\":\"Claude Code 작업이 완료되었습니다: $CLAUDE_TASK_SUMMARY\"}' $SLACK_WEBHOOK_URL",
            "description": "Slack 웹훅으로 완료 알림 전송",
            "timeout": 10000,
            "on_error": "warn"
          }
        ]
      }
    ],
    "Notification": [
      {
```

```
        "matcher": {
          "level": "error"
        },
        "hooks": [
          {
            "type": "command",
            "command": "curl -X POST -H 'Content-type: application/json' --data '{\"text\":\"   Claude Code 에러
발생: $CLAUDE_NOTIFICATION_MESSAGE\"}' $SLACK_WEBHOOK_URL",
            "description": "에러 발생 시 Slack 알림",
            "timeout": 10000,
            "on_error": "ignore"
          }
        ]
      }
    ]
  }
}
```

---

4. 테스트 자동 실행 (test-runner)

테스트 파일 또는 소스 파일 수정 시 관련 테스트를 자동 실행합니다.

```
{
  "$schema": "https://claude.ai/schemas/hooks.json",
  "description": "테스트 파일 수정 시 자동으로 해당 테스트를 실행하는 훅",
  "hooks": {
    "PostToolUse": [
      {
        "matcher": {
          "tool_name": "Edit|Write",
          "file_pattern": "**/*.test.{ts,tsx,js,jsx}|**/*.spec.{ts,tsx,js,jsx}"
        },
        "hooks": [
          {
            "type": "command",
            "command": "npm test -- --findRelatedTests $CLAUDE_FILE_PATH --passWithNoTests",
            "description": "수정된 테스트 파일 실행",
            "timeout": 60000,
            "on_error": "warn"
          }
        ]
      },
      {
        "matcher": {
          "tool_name": "Edit|Write",
          "file_pattern": "src/**/*.{ts,tsx,js,jsx}",
          "exclude_pattern": "**/*.test.*|**/*.spec.*"
        },
        "hooks": [
          {
            "type": "command",
            "command": "npm test -- --findRelatedTests $CLAUDE_FILE_PATH --passWithNoTests",
            "description": "소스 파일 수정 시 관련 테스트 실행",
            "timeout": 60000,
            "on_error": "warn"
          }
        ]
      }
    ]
  }
}
```

관련 문서

- MCP 서버 예제
- 모범 사례

예제: MCP 서버

MCP 서버 설정 예제

Model Context Protocol (MCP) 서버 설정 예제입니다.

MCP 개요

MCP는 Claude Code가 외부 시스템과 상호작용할 수 있게 해주는 프로토콜입니다.

설정 위치

```
~/.claude/settings.json
```

---

1. 데이터베이스 서버

PostgreSQL, SQLite, Redis 등 데이터베이스 연동 설정입니다.

```
{
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-postgres",
        "postgresql://user:password@localhost:5432/mydb"
      ],
      "description": "PostgreSQL 데이터베이스 연동",
      "env": {
        "PGPASSWORD": "${POSTGRES_PASSWORD}"
      }
    },
    "sqlite": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-sqlite",
        "--db-path",
        "./data/app.db"
      ],
      "description": "SQLite 로컬 데이터베이스"
    },
    "redis": {
      "command": "npx",
      "args": [
        "-y",
        "mcp-server-redis",
        "--url",
        "redis://localhost:6379"
      ],
```

```
      "description": "Redis 캐시 서버 연동"
    }
  }
}
```

---

2. 브라우저 자동화

Playwright, Puppeteer 등 브라우저 자동화 설정입니다.

```
{
  "mcpServers": {
    "playwright": {
      "command": "npx",
      "args": [
        "-y",
        "@anthropic/mcp-server-playwright"
      ],
      "description": "Playwright 브라우저 자동화",
      "env": {
        "PLAYWRIGHT_HEADLESS": "true"
      }
    },
    "puppeteer": {
      "command": "npx",
      "args": [
        "-y",
        "@anthropic/mcp-server-puppeteer"
      ],
      "description": "Puppeteer 브라우저 제어",
      "env": {
        "PUPPETEER_HEADLESS": "true"
      }
    },
    "chrome-devtools": {
      "command": "npx",
      "args": [
        "-y",
        "@anthropic/mcp-server-chrome-devtools"
      ],
      "description": "Chrome DevTools 연동"
    }
  }
}
```

---

3. 파일 시스템

파일 시스템, Git, 메모리 저장 등 설정입니다.

```
{
  "mcpServers": {
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "--allowed-directories",
        "/home/user/projects,/tmp"
```

```
      ],
      "description": "파일 시스템 확장 접근"
    },
    "git": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-git"
      ],
      "description": "Git 저장소 고급 기능"
    },
    "memory": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-memory"
      ],
      "description": "세션 간 메모리 저장"
    }
  }
}
```

---

4. 커스텀 API

GitHub, Slack, Notion, Linear 등 외부 API 연동 설정입니다.

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-github"
      ],
      "description": "GitHub API 연동",
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "slack": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-slack"
      ],
      "description": "Slack API 연동",
      "env": {
        "SLACK_BOT_TOKEN": "${SLACK_BOT_TOKEN}"
      }
    },
    "notion": {
      "command": "npx",
      "args": [
        "-y",
        "mcp-server-notion"
      ],
      "description": "Notion API 연동",
      "env": {
        "NOTION_API_KEY": "${NOTION_API_KEY}"
      }
    },
    "linear": {
      "command": "npx",
```

```
      "args": [
        "-y",
        "mcp-server-linear"
      ],
      "description": "Linear 이슈 트래커 연동",
      "env": {
        "LINEAR_API_KEY": "${LINEAR_API_KEY}"
      }
    }
  }
}
```

---

5. 전체 설정 예제

여러 MCP 서버를 조합한 전체 설정 예제입니다.

```
{
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-postgres",
        "postgresql://user:password@localhost:5432/mydb"
      ],
      "env": {
        "PGPASSWORD": "${POSTGRES_PASSWORD}"
      }
    },
    "playwright": {
      "command": "npx",
      "args": [
        "-y",
        "@anthropic/mcp-server-playwright"
      ],
      "env": {
        "PLAYWRIGHT_HEADLESS": "true"
      }
    },
    "github": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-github"
      ],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "--allowed-directories",
        "/home/user/projects"
      ]
    },
    "memory": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-memory"
```

```
        ]
      }
    },
    "permissions": {
      "allow": [
        "Bash(npm run *)",
        "Bash(git *)",
        "Read",
        "Write",
        "Edit"
      ],
      "deny": [
        "Bash(rm -rf /)",
        "Bash(sudo *)"
      ]
    },
    "preferences": {
      "autoApprove": false,
      "verboseOutput": true
    }
}
```

관련 문서

- Hooks 예제
- 모범 사례

예제: 프롬프트

효과적인 프롬프트 예제

Claude Code에서 사용할 수 있는 효과적인 프롬프트 패턴 모음입니다.

카테고리

카테고리	설명
코드 분석	코드 분석 및 이해
코드 작성	새로운 코드 작성
버그 수정	버그 분석 및 수정
리팩토링	코드 개선
테스트	테스트 코드 작성

프롬프트 작성 원칙

1. 구체적으로 작성

```
"코드 수정해줘"
"src/utils/date.ts의 formatDate 함수에서 타임존을 KST로 변환하는 로직을 추가해줘"
```

2. 컨텍스트 포함

```
"에러 수정해줘"
"npm run build 실행 시 다음 에러가 발생해: [에러 메시지]. 이 에러를 수정해줘"
```

3. 제약 조건 명시

```
"React 18의 새로운 훅을 사용하지 않고 구현해줘"
"외부 라이브러리 추가 없이 구현해줘"
```

4. 단계적 작업 요청

```
"먼저 현재 코드를 분석하고, 그 다음 리팩토링 계획을 세워줘"
```

관련 문서

- 핵심 원칙

- 모범 사례

프롬프트: 코드 분석

코드 분석 프롬프트

구조 분석

프로젝트 전체 구조

```
이 프로젝트의 전체 구조를 분석해줘.
주요 디렉토리와 파일들의 역할을 설명해줘.
```

특정 모듈 분석

```
src/services/auth/ 디렉토리의 인증 모듈을 분석해줘.
각 파일의 역할과 상호 의존성을 설명해줘.
```

코드 이해

함수 동작 설명

```
src/utils/parser.ts의 parseJSON 함수가 어떻게 동작하는지 설명해줘.
입력값과 출력값, 에러 처리 방식을 포함해서.
```

데이터 흐름 추적

```
사용자가 로그인 버튼을 클릭했을 때부터 토큰이 저장되기까지의
데이터 흐름을 추적해줘.
```

품질 분석

코드 품질 리뷰

```
src/components/Dashboard.tsx 파일의 코드 품질을 분석해줘.
개선이 필요한 부분과 그 이유를 설명해줘.
```

성능 분석

```
이 React 컴포넌트의 렌더링 성능에 영향을 줄 수 있는 부분을 찾아줘.
불필요한 리렌더링이 발생할 수 있는 곳을 식별해줘.
```

보안 취약점 분석

```
이 API 엔드포인트 코드에서 보안 취약점이 있는지 검토해줘.
SQL 인젝션, XSS, CSRF 등 OWASP Top 10 관점에서 분석해줘.
```

의존성 분석

패키지 의존성

```
package.json을 분석해서 사용하지 않는 의존성이 있는지 찾아줘.
업데이트가 필요한 패키지도 확인해줘.
```

모듈 의존성

```
src/services/ 디렉토리의 모듈 간 의존성 그래프를 분석해줘.
순환 의존성이 있는지 확인해줘.
```

관련 문서

- 코드 작성 프롬프트
- 코드 리뷰 워크플로

프롬프트: 코드 작성

코드 작성 프롬프트

새 기능 구현

API 엔드포인트

```
다음 요구사항에 맞는 REST API 엔드포인트를 구현해줘:
- 경로: POST /api/users/register
- 입력: { email, password, name }
- 검증: 이메일 형식, 비밀번호 8자 이상
- 응답: 생성된 사용자 정보 (비밀번호 제외)
- 에러: 중복 이메일 시 409 반환

기존 프로젝트의 코딩 스타일과 에러 처리 패턴을 따라줘.
```

React 컴포넌트

```
다음 요구사항의 React 컴포넌트를 구현해줘:
- 컴포넌트명: UserProfileCard
- Props: user (id, name, email, avatar)
- 기능: 프로필 이미지, 이름, 이메일 표시
- 스타일: 기존 프로젝트의 Tailwind 클래스 사용
- 접근성: 적절한 aria 속성 포함
```

유틸리티 함수

```
다음 유틸리티 함수를 구현해줘:
- 함수명: debounce
- 파라미터: func (함수), wait (밀리초)
- 반환: 디바운스된 함수
- 타입: 제네릭으로 원본 함수 타입 유지
- 추가: cancel 메서드 포함
```

기존 기능 확장

기능 추가

```
src/services/email.ts의 sendEmail 함수에 다음 기능을 추가해줘:
- 첨부파일 지원 (최대 5개, 각 10MB 이하)
- 재시도 로직 (최대 3회, 지수 백오프)
- 전송 결과 로깅

기존 인터페이스와 호환성을 유지해줘.
```

옵션 추가

```
formatDate 함수에 다음 옵션을 추가해줘:
- locale: 'ko-KR' | 'en-US' (기본값: 'ko-KR')
```

```
- includeTime: boolean (기본값: false)
- relative: boolean (기본값: false) - "3일 전" 형식

기존 호출부는 수정 없이 동작해야 해.
```

통합 작업

외부 API 연동

```
Stripe 결제 API를 연동하는 서비스를 구현해줘:
- 파일: src/services/payment.ts
- 기능: 결제 생성, 취소, 환불
- 환경변수: STRIPE_SECRET_KEY 사용
- 에러 처리: Stripe 에러를 커스텀 에러로 변환
- 테스트: 모킹 가능한 구조로 설계
```

데이터베이스 마이그레이션

```
users 테이블에 다음 컬럼을 추가하는 마이그레이션을 작성해줘:
- phone_number: varchar(20), nullable
- verified_at: timestamp, nullable
- preferences: jsonb, default {}

Prisma 마이그레이션 파일 형식으로 작성해줘.
```

관련 문서

- 코드 분석 프롬프트
- 테스트 프롬프트

프롬프트: 버그 수정

버그 수정 프롬프트

에러 분석

런타임 에러

```
다음 에러가 발생해:

TypeError: Cannot read property 'map' of undefined
    at UserList (src/components/UserList.tsx:15:23)
    at renderWithHooks (node_modules/react-dom/...)

이 에러의 원인을 분석하고 수정해줘.
```

빌드 에러

```
npm run build 실행 시 다음 에러가 발생해:

TS2345: Argument of type 'string | undefined' is not assignable
to parameter of type 'string'.

src/utils/api.ts:42:15 - error TS2345

원인을 분석하고 타입 안전하게 수정해줘.
```

테스트 실패

```
npm test 실행 시 다음 테스트가 실패해:

FAIL src/services/auth.test.ts
  ● AuthService > login > should return user on valid credentials

    expect(received).toEqual(expected)
    Expected: {"id": 1, "email": "test@test.com"}
    Received: undefined

테스트 코드와 실제 구현을 분석해서 문제를 수정해줘.
```

동작 버그

예상과 다른 동작

```
문제:
- 기대 동작: 로그인 후 대시보드로 리다이렉트
- 실제 동작: 로그인은 성공하지만 현재 페이지에 머무름

관련 파일:
- src/pages/Login.tsx
- src/hooks/useAuth.ts
- src/contexts/AuthContext.tsx
```

```
원인을 찾아서 수정해줘.
```

간헐적 버그

```
문제: 가끔 API 호출이 두 번 발생함
조건: React Strict Mode에서만 발생하는 것 같음
증상: 네트워크 탭에서 같은 요청이 두 번 보임

src/hooks/useFetch.ts를 분석해서 원인을 찾고 수정해줘.
```

성능 버그

느린 렌더링

```
Dashboard 페이지가 데이터 로딩 후 렌더링이 느려.
React DevTools Profiler에서 불필요한 리렌더링이 보임.

src/pages/Dashboard.tsx를 분석하고 최적화해줘.
memo, useMemo, useCallback 적용이 필요한 부분을 찾아줘.
```

메모리 누수

```
SPA에서 페이지 이동 시 메모리가 계속 증가해.
Chrome DevTools 메모리 탭에서 detached DOM nodes가 증가함.

컴포넌트의 cleanup 로직을 검토하고 메모리 누수를 수정해줘.
```

호환성 버그

브라우저 호환성

```
Safari에서 날짜 파싱이 안 돼:
new Date('2024-01-15') → Invalid Date

Safari 호환되도록 수정해줘.
```

모바일 이슈

```
모바일에서 드롭다운 메뉴가 화면 밖으로 나가.
viewport를 고려해서 메뉴 위치를 조정하는 로직을 추가해줘.
```

관련 문서

- 디버깅 워크플로
- 코드 분석 프롬프트

프롬프트: 리팩토링

리팩토링 프롬프트

코드 구조 개선

함수 분리

```
src/services/order.ts의 processOrder 함수가 200줄이 넘어.
다음 원칙에 따라 작은 함수들로 분리해줘:
- 단일 책임 원칙
- 각 함수는 50줄 이내
- 함수명은 동작을 명확히 표현
- 기존 동작은 그대로 유지

먼저 분리 계획을 세우고, 단계적으로 진행해줘.
```

중복 제거

```
src/components/ 디렉토리에서 비슷한 패턴의 코드가 반복되고 있어:
- UserCard, ProductCard, OrderCard가 거의 동일한 구조
- 각각 다른 데이터 타입만 사용

공통 컴포넌트로 추출하고, 제네릭으로 타입 안전성을 유지해줘.
```

조건문 단순화

```
src/utils/permissions.ts의 checkPermission 함수가 복잡해:
- 중첩된 if문이 5단계
- 조건이 뒤엉켜 있음

early return 패턴과 guard clause를 적용해서 단순화해줘.
```

패턴 적용

디자인 패턴 도입

```
src/services/notification.ts에 Strategy 패턴을 적용해줘:
- 현재: if-else로 채널별 분기 (email, sms, push)
- 목표: 각 채널을 별도 전략 클래스로 분리
- 새 채널 추가가 쉬워야 함
```

Repository 패턴

```
데이터 접근 로직을 Repository 패턴으로 리팩토링해줘:
- 현재: 컴포넌트에서 직접 API 호출
- 목표: Repository 레이어 분리
- 테스트 시 쉽게 모킹 가능하도록
```

타입 개선

any 제거

```
src/ 디렉토리에서 any 타입을 사용하는 곳을 찾아서
구체적인 타입으로 변경해줘.

우선순위:
1. 함수 파라미터
2. 반환 타입
3. 변수 선언
```

타입 정의 개선

```
src/types/api.ts의 타입 정의를 개선해줘:
- 중복된 타입 통합
- 유니온 타입으로 상태 표현
- 유틸리티 타입 활용 (Pick, Omit, Partial)
```

성능 최적화

불필요한 연산 제거

```
이 코드의 불필요한 연산을 최적화해줘:
- 반복문 내 불변값 캐싱
- 조기 종료 조건 추가
- 불필요한 배열 복사 제거
```

메모이제이션 적용

```
Dashboard 컴포넌트에 적절한 메모이제이션을 적용해줘:
- useMemo: 계산 비용이 큰 값
- useCallback: props로 전달되는 함수
- memo: 자주 리렌더링되는 자식 컴포넌트
```

관련 문서

- 리팩토링 워크플로
- 코드 분석 프롬프트

프롬프트: 테스트

테스트 작성 프롬프트

단위 테스트

유틸리티 함수 테스트

```
src/utils/validation.ts의 validateEmail 함수에 대한 테스트를 작성해줘:
- 유효한 이메일 케이스 (일반, 서브도메인, 특수문자)
- 무효한 이메일 케이스 (@ 없음, 도메인 없음, 공백)
- 엣지 케이스 (빈 문자열, null, undefined)
- describe/it 구조로 그룹화
```

서비스 클래스 테스트

```
src/services/UserService.ts의 테스트를 작성해줘:
- 외부 의존성(API, DB)은 모킹
- 성공/실패 케이스 모두 커버
- 에러 핸들링 검증
- 파라미터 검증 로직 테스트
```

통합 테스트

API 엔드포인트 테스트

```
POST /api/auth/login 엔드포인트의 통합 테스트를 작성해줘:
- 성공: 유효한 자격 증명으로 토큰 반환
- 실패: 잘못된 비밀번호 (401)
- 실패: 존재하지 않는 사용자 (404)
- 실패: 잘못된 입력 형식 (400)
- 레이트 리밋 검증

supertest 사용, 테스트 DB로 분리.
```

데이터베이스 통합 테스트

```
UserRepository의 통합 테스트를 작성해줘:
- 실제 테스트 DB 사용
- 각 테스트 전후 데이터 정리
- CRUD 전체 사이클 테스트
- 트랜잭션 롤백 테스트
```

컴포넌트 테스트

React 컴포넌트 테스트

```
src/components/LoginForm.tsx의 테스트를 작성해줘:
- @testing-library/react 사용
- 렌더링 테스트
- 사용자 입력 시뮬레이션
- 폼 제출 테스트
- 에러 메시지 표시 테스트
- 로딩 상태 테스트
```

훅 테스트

```
src/hooks/useAuth.ts의 테스트를 작성해줘:
- @testing-library/react-hooks 사용
- 초기 상태 검증
- 로그인/로그아웃 동작 테스트
- 토큰 만료 처리 테스트
- 컨텍스트 의존성 모킹
```

E2E 테스트

사용자 플로우 테스트

```
Playwright로 로그인 플로우 E2E 테스트를 작성해줘:

1. 로그인 페이지 접속
2. 이메일/비밀번호 입력
3. 로그인 버튼 클릭
4. 대시보드 리다이렉트 확인
5. 사용자 정보 표시 확인

실패 케이스도 포함해줘.
```

크리티컬 패스 테스트

```
결제 플로우의 E2E 테스트를 작성해줘:
- 상품 선택 → 장바구니 → 결제 → 확인
- 각 단계의 UI 상태 검증
- 에러 시 롤백 확인
- 스크린샷 캡처 포함
```

테스트 개선

커버리지 향상

```
현재 테스트 커버리지 리포트를 분석하고,
커버되지 않은 src/services/ 코드에 대한 테스트를 추가해줘.
분기 커버리지 80% 이상 목표.
```

플레이키 테스트 수정

```
CI에서 가끔 실패하는 테스트가 있어:
- src/components/DataTable.test.tsx

비동기 타이밍 이슈인 것 같아.
waitFor, findBy 등을 적절히 사용해서 안정화해줘.
```

관련 문서

- 코드 작성 프롬프트
- 디버깅 워크플로

템플릿: CLAUDE.md

CLAUDE.md 템플릿

프로젝트 루트에 배치하여 Claude Code에 프로젝트 컨텍스트를 제공하는 파일입니다.

사용법

- 아래 템플릿을 복사합니다
- 프로젝트 루트에 CLAUDE.md 파일로 저장합니다
- 프로젝트에 맞게 내용을 수정합니다

템플릿

````
# 프로젝트명

> 프로젝트 한 줄 설명

## 개요

프로젝트에 대한 간단한 설명을 작성합니다.

## 기술 스택

- **언어**: TypeScript 5.x
- **프레임워크**: React 18 / Next.js 14
- **데이터베이스**: PostgreSQL
- **인프라**: Docker, AWS

## 프로젝트 구조

```
src/
├── components/    # UI 컴포넌트
├── services/      # 비즈니스 로직
├── utils/         # 유틸리티 함수
├── types/         # 타입 정의
└── hooks/         # 커스텀 훅
```

## 빌드 및 실행

```bash
# 의존성 설치
npm install

# 개발 서버
npm run dev

# 빌드
npm run build
````
````

```
# 테스트
npm test

# 린트
npm run lint
```

코딩 컨벤션

- ESLint + Prettier 사용
- 함수형 컴포넌트 선호
- 파일명: kebab-case
- 컴포넌트명: PascalCase
- 함수/변수명: camelCase

환경 변수

```bash
# .env.example
DATABASE_URL=postgresql://...
API_KEY=your-api-key
```

주요 기능

1. **기능1**: 설명
2. **기능2**: 설명
3. **기능3**: 설명

주의사항

- 프로덕션 DB에 직접 쿼리 금지
- API 키 하드코딩 금지
- `.env` 파일 커밋 금지

참고 문서

- [API 문서](docs/api.md)
- [아키텍처 결정 기록](docs/adr/)

섹션별 설명

섹션	설명
프로젝트명	프로젝트 이름과 한 줄 설명
개요	프로젝트의 목적과 주요 기능
기술 스택	사용 중인 언어, 프레임워크, 도구
프로젝트 구조	디렉토리 구조와 각 폴더의 역할
빌드 및 실행	자주 사용하는 명령어
코딩 컨벤션	코드 스타일 규칙
환경 변수	필요한 환경 변수 목록
주요 기능	핵심 기능 목록

섹션	설명
주의사항	개발 시 주의할 점
참고 문서	추가 문서 링크

관련 문서

- 핵심 원칙
- 시작하기

기여 가이드

기여 가이드

Claude Code Workflow 프로젝트에 기여해 주셔서 감사합니다! 이 문서는 프로젝트에 기여하는 방법을 안내합니다.

기여 방법

1. Issue 생성

기여를 시작하기 전에 Issue를 먼저 생성해 주세요. 이렇게 하면:

- 중복 작업을 방지할 수 있습니다
- 메인테이너의 피드백을 미리 받을 수 있습니다
- 다른 기여자들과 협업할 수 있습니다

Issue Templates

프로젝트에서 제공하는 템플릿을 사용해 주세요:

템플릿	용도	상세 가이드
버그 리포트	오류, 버그 신고	Issue Templates 가이드
기능 요청	새 기능, 개선 제안	Issue Templates 가이드
문서 개선	문서 수정/추가	Issue Templates 가이드
워크플로 제안	새 워크플로 패턴	Issue Templates 가이드
질문	사용법 질문	Issue Templates 가이드

2. Fork & Clone

```
# 저장소 Fork (GitHub UI에서)

# Clone
git clone https://github.com/YOUR_USERNAME/claude-code-workflow.git
cd claude-code-workflow
```

3. Branch 생성

```
# 기능 브랜치 생성
git checkout -b feature/amazing-feature

# 또는 버그 수정
git checkout -b fix/bug-description

# 또는 문서 수정
git checkout -b docs/update-description
```

4. 변경사항 작성

프로젝트 구조를 참고하여 적절한 위치에 변경사항을 작성합니다:

```
claude-code-workflow/
├─── docs/               # 핵심 문서
├─── workflows/          # 워크플로 패턴
├─── examples/           # 예제
│    ├─── hooks/
│    ├─── mcp/
│    └─── prompts/
└─── templates/          # 템플릿
```

5. Commit

```
# 변경사항 스테이징
git add .

# 커밋 (컨벤션에 맞게)
git commit -m "feat: Add amazing feature"
```

Commit 메시지 컨벤션

타입	설명	예시
feat	새 기능	feat: Add testing workflow
fix	버그 수정	fix: Correct typo in principles.md
docs	문서 수정	docs: Update getting-started guide
refactor	리팩토링	refactor: Reorganize examples folder
chore	기타 작업	chore: Update issue templates

6. Push & PR

```
# Push
git push origin feature/amazing-feature
```

GitHub에서 Pull Request를 생성합니다.

Labels

프로젝트에서 사용하는 Labels 체계입니다:

그룹	접두사	용도
카테고리	category:	docs, workflow, example, template, principle
영역	area:	hooks, mcp, prompts, code-review, debugging, refactoring
타입	type:	enhancement, bug, question, research
우선순위	priority:	high, medium, low

자세한 내용은 Labels 가이드를 참조하세요.

PR 체크리스트

PR을 제출하기 전에 확인해 주세요:

- [ ] 관련 Issue가 있으면 PR에 연결했나요?
- [ ] Commit 메시지가 컨벤션을 따르나요?
- [ ] 문서를 수정한 경우 오타가 없나요?
- [ ] 예제 코드가 올바르게 동작하나요?
- [ ] 기존 문서와 일관된 스타일인가요?

기여 유형

Good First Issues

처음 기여하시는 분들을 위한 이슈입니다:

- good first issue 라벨이 붙은 이슈를 확인하세요
- 주로 오타 수정, 간단한 문서 개선 등이 해당됩니다

문서 기여

- 오타/문법 수정
- 설명 보완

- 예제 추가
- 번역

콘텐츠 기여

- 새 워크플로 패턴 추가
- 새 예제 추가
- 프롬프트 패턴 추가
- 템플릿 개선

도움이 필요하신가요?

- GitHub Discussions: 일반적인 질문이나 아이디어 공유
- Issue: 구체적인 버그나 기능 요청
- Wiki: 문서 확인

감사합니다!

저장소 구조

저장소 구조

이 문서는 Claude Code Workflow 저장소의 전체 구조와 각 파일의 역할을 설명합니다.

전체 구조

```
claude-code-workflow/
├── .github/                    # GitHub 설정
│   ├── CODEOWNERS              # 코드 소유자
│   ├── dependabot.yml          # 의존성 자동 업데이트
│   ├── FUNDING.yml             # 후원 설정
│   ├── labeler.yml             # PR 자동 라벨링 규칙
│   ├── ISSUE_TEMPLATE/         # Issue 템플릿 (6개)
│   ├── PULL_REQUEST_TEMPLATE.md
│   └── workflows/              # GitHub Actions (5개)
├── .editorconfig               # 에디터 설정
├── .gitignore                  # Git 무시 파일
├── .markdownlint.json          # 마크다운 린트 규칙
├── CODE_OF_CONDUCT.md          # 행동 강령
├── CONTRIBUTING.md             # 기여 가이드
├── LICENSE                     # MIT 라이선스
├── README.md                   # 프로젝트 소개
├── SECURITY.md                 # 보안 정책
├── SUPPORT.md                  # 지원 안내
├── docs/                       # 핵심 문서
├── examples/                   # 예제
│   ├── hooks/
│   ├── mcp/
│   └── prompts/
├── templates/                  # 템플릿
└── workflows/                  # 워크플로 패턴
```

루트 파일

커뮤니티 표준 파일

파일	설명	상세
README.md	프로젝트 소개	Badge, 목차, 시작 가이드
CONTRIBUTING.md	기여 가이드	Fork, PR, 커밋 컨벤션
CODE_OF_CONDUCT.md	행동 강령	Contributor Covenant 2.1
SECURITY.md	보안 정책	취약점 신고 방법
SUPPORT.md	지원 안내	도움 받는 방법, FAQ

파일	설명	상세
LICENSE	라이선스	MIT License

설정 파일

파일	설명
.gitignore	Git에서 무시할 파일 (OS, IDE, Node.js, Python 등)
.editorconfig	에디터 일관성 설정 (인코딩, 들여쓰기, 줄 끝)
.markdownlint.json	마크다운 린트 규칙

.github 디렉토리

Issue Templates

파일	아이콘	용도	자동 Label
bug_report.yml		버그 리포트	type: bug
feature_request.yml		기능 요청	type: enhancement
docs_improvement.yml		문서 개선	category: docs
workflow_proposal.yml		워크플로 제안	category: workflow, type: research
question.yml		질문	type: question
config.yml		설정	빈 이슈 비활성화, 외부 링크

기타 설정 파일

파일	설명
PULL_REQUEST_TEMPLATE.md	PR 생성 시 자동 적용되는 템플릿
CODEOWNERS	코드 소유자 지정 (자동 리뷰어 할당)
FUNDING.yml	GitHub Sponsors 후원 버튼
dependabot.yml	의존성 자동 업데이트 설정
labeler.yml	PR 경로 기반 자동 라벨링 규칙

GitHub Actions Workflows

워크플로	파일	트리거	기능
Markdown Lint	markdown-lint.yml	Push/PR (*.md)	마크다운 문법/스타일 검사
Link Check	link-check.yml	Push/PR, 매주 월요일	깨진 링크 자동 검사
Auto Label	auto-label.yml	PR Open	경로 기반 자동 라벨링
Welcome	welcome.yml	Issue/PR Open	첫 기여자 환영 메시지
Stale	stale.yml	매일 09:00 (KST)	오래된 이슈/PR 관리

GitHub Labels

카테고리 (category:)

Label	색상	설명
category: docs	파랑	문서 관련
category: workflow	보라	워크플로 패턴
category: example	청록	예제 코드
category: template	하늘	템플릿
category: principle	초록	핵심 원칙

영역 (area:)

Label	설명
area: hooks	Hooks 관련
area: mcp	MCP 서버 관련
area: prompts	프롬프트 관련
area: code-review	코드 리뷰 워크플로
area: debugging	디버깅 워크플로
area: refactoring	리팩토링 워크플로

타입 (type:)

Label	색상	설명
type: enhancement	시안	새 기능, 개선
type: bug	빨강	버그
type: question	분홍	질문
type: research	연초록	연구, 조사

우선순위 (priority:)

Label	색상	설명
priority: high	빨강	긴급
priority: medium	노랑	중간
priority: low	초록	낮음

기타

Label	설명
good first issue	초보자에게 좋은 이슈
help wanted	도움이 필요한 이슈
duplicate	중복된 이슈
wontfix	수정하지 않을 이슈

콘텐츠 디렉토리

docs/

핵심 문서가 위치합니다.

파일	설명
getting-started.md	시작 가이드
principles.md	핵심 원칙
best-practices.md	모범 사례

workflows/

워크플로 패턴 문서입니다.

파일	설명
code-review.md	코드 리뷰 워크플로
debugging.md	디버깅 워크플로
refactoring.md	리팩토링 워크플로

examples/

예제 코드 및 설정입니다.

디렉토리	설명
hooks/	Claude Code Hooks 예제
mcp/	MCP 서버 설정 예제
prompts/	효과적인 프롬프트 예제

templates/

프로젝트에서 바로 사용할 수 있는 템플릿입니다.

파일	설명
CLAUDE.md	프로젝트별 CLAUDE.md 템플릿

관련 문서

- 기여 가이드
- Labels 가이드
- Issue Templates
- Pull Request 가이드